AI-powered FRA Atlas

SRMIST SIH2025-IDEA-Presentation-...
Presentation

Problem Statement Title
Development of AI-powered FRA Atlas and WebGIS-based Decision Support System (DSS) for Integrated Monitoring of Forest Rights Act (FRA) Implementation.
(States to be concentrated: Madhya Pradesh, Tripura , Odisha, Telangana)

Problem Background :
The Forest Rights Act (FRA), 2006 recognizes the rights of forest-dwelling communities over land and forest resources. However, significant challenges persist:
• Legacy records of Individual Forest Rights (IFR), Community Rights (CR), and Community Forest Resource Rights (CFR) are scattered, non-digitized, and difficult to verify.
• There is no centralized, real-time visual repository (e.g., an FRA Atlas) of FRA claims and granted titles.
• Integration of satellite-based asset mapping (land, water bodies, farms, etc.) with FRA data is missing.
• Integration of legacy data with FRA Atlas is missing.
• Decision-makers lack a Decision Support System (DSS) to layer Central Sector Schemes (CSS) benefits (e.g., PM-KISAN, Jal Jeevan Mission, MGNREGA, DAJGUA (3 ministries)) for FRA patta holders.

Project Objectives
1. Digitize and standardize legacy data of FRA claims, verifications, and pattas, and integrate with FRA Atlas. FRA patta holders' shapefiles to be integrated.
2. Create an FRA Atlas showing potential and granted FRA areas using AI and satellite data.
3. Integrate a WebGIS portal to visualize and manage spatial and socio-economic data.
4. Use Remote Sensing and AI/ML to map capital and social assets (ponds, farms, forest resources) of FRA-holding villages.
5. Build a Decision Support System (DSS) to

recommend and layer CSS schemes based on mapped data, enhancing targeted development.

AI & Tech Components

1. Data Digitization
• Use suitable models to extract and standardize text from scanned FRA documents.
• Apply Named Entity Recognition (NER) to identify village names, patta holders, coordinates, and claim status.

2. AI-based Asset Mapping
• Employ Computer Vision on high-resolution satellite imagery to detect:
  • Agricultural land
  • Forest cover
  • Water bodies (ponds, streams)
  • Homesteads
• Classify land-use using supervised ML models (e.g., Random Forest, CNN).
• Add layers of information with respect to forest data, groundwater data, and infrastructure data (e.g., PM Gati Shakti).

3. WebGIS Integration
• Interactive layers (IFR/CR, village boundaries, land-use, assets)
• Filters by state/district/village/tribal group
• FRA progress tracking (village/block/district/state level)

4. Decision Support System (DSS)
• Build a rule-based + AI-enhanced DSS engine that:
  • Cross-links FRA holders with eligibility for CSS schemes such as DAJGUA and others.
  • Prioritizes interventions (e.g., borewells under Jal Shakti for villages with low water index).

Deliverables

1. AI-processed digital archive of FRA claims & decisions.
2. An interactive FRA Atlas on a WebGIS platform.

3. AI-generated asset maps for all FRA villages.
4. A DSS engine for scheme layering and policy formulation.

Target Users
• Ministry of Tribal Affairs
• District-level Tribal Welfare Departments & Line Departments of DAJGUA
• Forest and Revenue Departments
• Planning & Development Authorities
• NGOs working with tribal communities

Future Scope
• Incorporate real-time satellite feeds for monitoring CFR forests.
• Integrate IoT sensors for soil health, water quality, etc., in FRA lands (if feasible).
• Enable mobile-based feedback and updates from patta holders themselves.

give me solution for it, also give content according to the ppt uploaded

Got it 👍 I'll structure your solution content exactly according to the **SIH2025 PPT template** you uploaded.
Here's the slide-wise draft content for your **Problem Statement: AI-powered FRA Atlas and WebGIS-based DSS**.

Slide 1 – Title Slide

- **TEAM NAME**: *(Your Team Name)*
- **SMART INDIA HACKATHON 2025**
- **Problem Statement ID**: *(Insert ID)*
- **Problem Statement Title**: *Development of AI-powered FRA Atlas and WebGIS-based Decision Support System (DSS) for Integrated Monitoring of Forest Rights Act (FRA) Implementation*
- **Theme**: Agriculture & Rural Development / Governance
- **Category**: Software

Slide 2 – Idea Title & Proposed Solution

Idea Title: *AI-Powered FRA Atlas & WebGIS DSS for Tribal Empowerment*

Proposed Solution

- Centralized **FRA Atlas** digitizing IFR, CR, CFR records.
- **AI/ML for asset mapping**: detect farms, ponds, forests, homesteads from satellite imagery.
- **WebGIS portal** with interactive layers (village, forest, patta holders, assets).
- **Decision Support System (DSS)** to recommend & layer CSS schemes for FRA patta holders.

Innovation & Uniqueness

- First **AI-driven integration** of FRA records + satellite mapping + DSS.
- Enables **real-time monitoring & scheme targeting**.
- Empowers **tribal communities** through transparent decision-making.

Slide 3 – Technical Approach

Technologies Used

- **AI/ML**: CNN, Random Forest for land-use classification
- **OCR + NLP**: NER for digitizing FRA claims
- **Remote Sensing**: Sentinel-2, PlanetScope satellite imagery
- **WebGIS**: GeoServer, Leaflet/Mapbox, PostgreSQL + PostGIS
- **Backend**: Python/Django, Node.js
- **Frontend**: React + GIS libraries

Methodology

1. **Data Digitization**: OCR → NER → Standardized database
2. **Asset Mapping**: CV/ML on satellite data → classification layers
3. **Integration**: WebGIS with interactive filtering & dashboards
4. **DSS Engine**: Rule-based + AI-enhanced scheme recommendations
5. **Deployment**: Cloud-based (AWS/GCP) with APIs for integration

---

Slide 4 – Feasibility & Viability

Feasibility

- Availability of FRA legacy documents, satellite imagery & govt scheme datasets.
- Proven AI/ML models for land classification & OCR.

Potential Challenges

- Non-uniform legacy records & data gaps
- High-resolution satellite imagery cost
- Adoption by local departments

Strategies to Overcome

- Use **open-source data** (Sentinel, Bhuvan, FSI maps).
- Data cleaning pipelines + active learning for OCR accuracy.
- Training workshops for district & tribal welfare officials.

Slide 5 – Impact & Benefits

Impact on Target Users

- Ministry of Tribal Affairs & State Departments → **data-driven monitoring**
- Tribal Communities → **direct benefit through scheme access**
- NGOs/Planners → **real-time insights** for interventions

Benefits

- **Social**: Empowerment of FRA patta holders, tribal inclusion
- **Economic**: Optimized scheme allocation, improved livelihoods
- **Environmental**: Sustainable land & forest resource management
- **Governance**: Transparent, accountable, real-time decision-making

---

Slide 6 – Research & References + Team Members

Research & References

- Forest Rights Act, 2006
- National Remote Sensing Centre (NRSC) Bhuvan Portal
- UN FAO Land Use Classification Guidelines
- MoTA Annual FRA Reports
- PM Gati Shakti & CSS scheme documents

Team Members

- Name 1 – Role (AI/ML)
- Name 2 – Role (GIS & Remote Sensing)

- Name 3 – Role (Web Development)
- Name 4 – Role (Domain/Documentation)
- Name 5 – Role (Integration & Deployment)

✅ This keeps it within the **6-slide SIH limit** and in the **exact format required**.

Do you want me to **fill this directly into your PPT file** (so you get a ready-to-submit version in PDF), or do you prefer keeping it as text so you can tweak content yourself?

suppose i want to procede like this,
Solution Workflow

1. Data Collection & Digitization

Scan and OCR FRA documents (claims, pattas, verification reports).

Use NLP + NER to extract patta holder names, village, land extent, claim type.

Create a standardized FRA database.

2. AI-Powered FRA Atlas Creation

Use Remote Sensing + AI/ML for asset detection:

CNN for land-use classification (agriculture, water bodies, forest cover).

CV models for detecting ponds, farms, homesteads.

Integrate FRA shapefiles with satellite-based maps.

3. WebGIS Integration

Develop a WebGIS Portal with interactive layers:

FRA claims (IFR/CR/CFR)

Asset maps

Socio-economic overlays (CSS scheme data, infrastructure, groundwater).

Provide filters (state/district/village/tribal group).

4. Decision Support System (DSS)

Rule-based + AI-driven engine:

Match FRA holders with eligible Central Schemes (PM-KISAN, Jal Shakti, MGNREGA).

Generate recommendations (e.g., borewells in low-water zones).

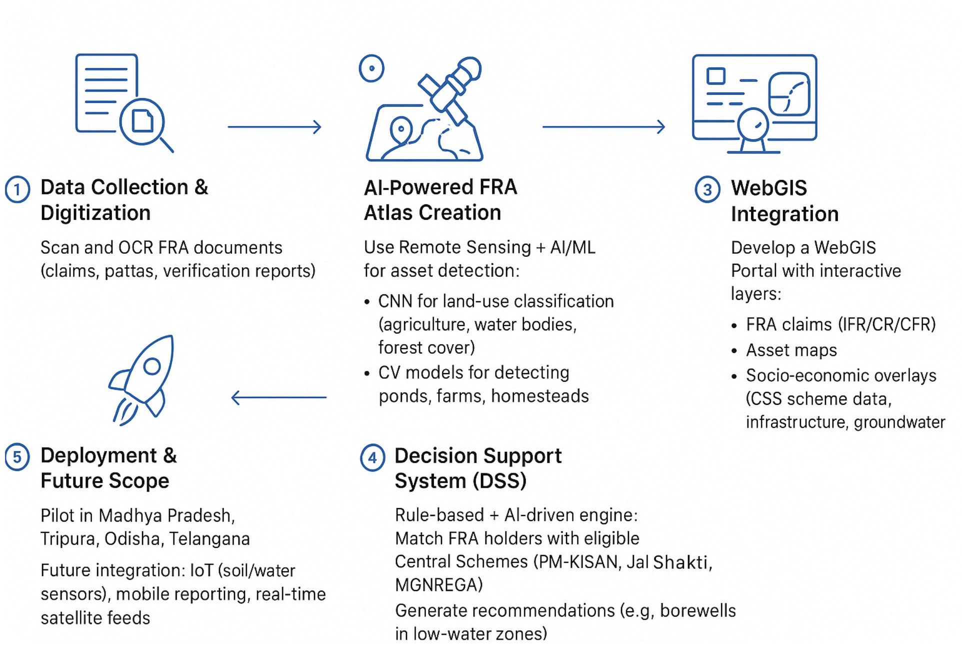
Support policy-level dashboards.

5. Deployment & Future Scope

Pilot in Madhya Pradesh, Tripura, Odisha, Telangana.

Future integration: IoT (soil/water sensors), mobile reporting, real-time satellite feeds.

generate a architecture and workflow diagram for this



How it addresses the problem

Here's how your **proposed solution addresses the given problem statement** (you can directly use this for the PPT slide on "Proposed Solution" → *How it addresses the problem*):

How It Addresses the Problem

- **Fragmented & Non-digitized FRA Records → Digitization & NLP**
  Scanned FRA claims, pattas, and reports are digitized using OCR + NER, creating a **centralized structured database** accessible for monitoring.
- **Lack of Centralized FRA Atlas → AI-Powered Atlas**
  Integration of **satellite imagery + FRA shapefiles** builds an **interactive FRA Atlas** showing potential and granted areas in real time.
- **No Integration with Assets → AI-Based Asset Mapping**
  Remote sensing + ML classify **agriculture, forests, water bodies, homesteads**, providing detailed **village-level asset maps** linked to FRA holdings.
- **No DSS for Decision-Making → AI-driven DSS Engine**
  The DSS cross-links **FRA holders with eligibility under Central Sector Schemes (CSS)** and provides **recommendations** (e.g., irrigation for low-water areas).
- **Disconnected Legacy Data → WebGIS Visualization**
  All digitized claims, asset layers, and socio-economic overlays are accessible on a

WebGIS platform with filters (state/district/village/tribe).

- **Future Monitoring Challenges → Scalable & Extendable Design**
  Supports **IoT sensors, mobile-based updates, and real-time satellite feeds** for continuous monitoring and feedback.

Do you want me to **convert this into concise bullet points** (ready for PPT formatting) or keep it as an explanatory block?

keep it short and crisp according to slide number 2

Got it 👍 Here's the **short & crisp version for Slide 2 (Proposed Solution → How it addresses the problem):**

How It Addresses the Problem

- Converts **scattered FRA records** into a **centralized digital database**.
- Builds an **AI-powered FRA Atlas** integrating shapefiles + satellite data.
- Maps **land, water, forest, and assets** using AI/ML.
- Provides a **WebGIS portal** with interactive filters & progress tracking.
- **DSS engine links FRA holders** with eligible govt schemes for targeted benefits.

Would you like me to also make a **crisp version for "Innovation & Uniqueness"** (same slide) so the slide looks balanced?

yes

Perfect 👍 Here's the **short & crisp version for "Innovation & Uniqueness"** (to go along with Slide 2):

Innovation & Uniqueness

- **First-of-its-kind integration** of FRA data + satellite asset mapping + DSS.
- **AI-driven insights** for land-use & asset identification at village level.

- **Dynamic WebGIS Atlas** with real-time monitoring capabilities.
- **Data-to-policy bridge**: Direct linkage of FRA holders with govt schemes.
- **Scalable model** adaptable for other states & community rights programs.

Do you want me to now **combine Slide 2 full content (Idea Title + Proposed Solution + How it addresses the problem + Innovation & Uniqueness)** into a ready-to-insert version for your PPT?

Forest Rights Act, 2006
National Remote Sensing Centre (NRSC) Bhuvan Portal
UN FAO Land Use Classification Guidelines
MoTA Annual FRA Reports
PM Gati Shakti & CSS scheme documents

give me links for this

Absolutely! Here are authoritative links for each of your key references—perfect for including in your presentation or documentation:

Reference Links

- **Forest Rights Act, 2006**
  Full text (PDF) of the Act *"Scheduled Tribes and Other Traditional Forest Dwellers (Recognition of Forest Rights) Act, 2006"* → India Code FAOLEX
- **National Remote Sensing Centre (NRSC) – Bhuvan Portal**
  The official Bhuvan GIS portal by ISRO/NRSC (Indian geospatial platform with satellite data and thematic mapping) → Bhuvan Wikipedia
- **UN-FAO Land Cover Classification Guidelines**
  The Land Cover Classification System and associated guidelines/manual for standardized land cover mapping → Open Knowledge FAO Academia
- **MoTA Annual FRA Reports**
  Portal with annual reports and scheme info from the Ministry of Tribal Affairs (includes FRA-related materials) → India.gov.in +1
- **PM Gati-Shakti & CSS Scheme Documents**
  a) Overview of the PM-Gati-Shakti National Master Plan → India.gov.in

pmgatishakti.gov.in

b) Detailed Compendium and infrastructure planning documents related to PM-Gati-Shakti → Press Information Bureau pmgatishakti.gov.in

---

Let me know if you'd like these formatted with titles and clickable links for your PPT or report!

Sources

For now i have to create a prototype in 5-6 days, how to start working for it, what tools, frameworks are required, which libraries and extenions need to be installed, give me complete step by step guide for this. Keeping in mind, i don't know anything about any things, but i want all to be integrated together and my prototype should be working

Awesome — you can pull a working prototype together in 5–6 days. Below I'll give you a **beginner-friendly, day-by-day plan**, exact tools to install, commands (macOS + Ubuntu), and **copy-paste code snippets** you can run right away. I'll prioritize a minimal, integrated MVP that **demonstrates all key pieces end-to-end**: OCR → structured DB → simple asset detection (water/veg) → WebGIS frontend → rule-based DSS.

> Note: I assume you're on a desktop (macOS/Linux). I show Homebrew commands for macOS and apt for Ubuntu when relevant. If you want Windows-specific commands later I'll add them.

---

1) Minimal MVP (what must work in 5–6 days)

- Upload scanned FRA images → OCR + NER → store structured records.
- Ingest one Sentinel-2 scene (or use Bhuvan sample) → compute NDWI/NDVI → derive water/vegetation masks → vectorize to GeoJSON.
- Serve FRA GeoJSON + asset GeoJSON from a simple Flask backend (PostGIS optional for MVP).
- Web map (Leaflet) showing layers, popups, simple filters.

- Small DSS: a rule-based function that recommends schemes (example rules) and a simple dashboard view.

---

2) Key tools (install these first)

- **Python 3.11+** (for backend, OCR, raster processing). Official download/install. Python.org
- **Postgres + PostGIS** (spatial DB; optional for MVP but recommended). PostGIS
- **Tesseract OCR** (OCR engine) — required for offline OCR. Tesseract OCR
- **GDAL / rasterio / geopandas** (for geospatial raster/vector processing).
- **Leaflet** (front-end mapping library) — simple to use and light-weight. leafletjs.com
- **Satellite data source**: Copernicus / Sentinel-2 or Bhuvan for Indian data. (free access). Copernicus Data Space Ecosystem bhuvan.nrsc.gov.in

(Installing these gives you everything you need for the prototype.)

---

3) Quick install checklist (macOS via Homebrew — fastest for beginners)

Open Terminal.

Install Homebrew (if not already):

```bash
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install Python, GDAL, PostgreSQL/PostGIS, Tesseract:

```bash
brew install python gdal postgresql postgis tesseract
```

Start PostgreSQL:

```bash
brew services start postgresql
createdb fra_db
```

```
psql fra_db -c "CREATE EXTENSION postgis;"
```

Create Python venv and install libs:

```bash
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install flask geopandas rasterio shapely pillow pytesseract spacy scikit-image scikit-learn
sqlalchemy psycopg2-binary
python -m spacy download en_core_web_sm
```

(If rasterio fails, make sure `brew install gdal` completed successfully before `pip install rasterio`.)

Ubuntu (apt) equivalents (if on Ubuntu):

```bash
sudo apt update
sudo apt install python3 python3-venv python3-pip gdal-bin libgdal-dev postgresql
postgresql-contrib postgis tesseract-ocr
# create db + postgis in psql shell
sudo -u postgres createdb fra_db
sudo -u postgres psql -c "CREATE EXTENSION postgis;" -d fra_db
# Python venv
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install flask geopandas rasterio shapely pillow pytesseract spacy scikit-image scikit-learn
sqlalchemy psycopg2-binary
python -m spacy download en_core_web_sm
```

---

4) Day-by-day 6-day plan (deliverable at end of each day)

Day 0 (setup — half day)

- Install software above. Verify `python --version`, `psql --version`, `tesseract --version`.

- Create a GitHub repo and a folder structure:

```kotlin
project/
  backend/
  frontend/
  data/
  notebooks/
  docs/
```

Day 1 — OCR pipeline + DB (deliverable: simple OCR upload endpoint)

- Build a tiny Flask app to accept uploaded scanned images/PDF and run Tesseract OCR.
- Run spaCy NER on OCR text to extract entities (village, names) — for prototype use English; treat NER as demo.
- Save extracted records to a simple table (start with SQLite or Postgres).
- **Deliverable:** `/upload` endpoint that returns extracted text + entities.

Minimal Flask OCR snippet (copy & run):

```python
# backend/app.py
from flask import Flask, request, jsonify
from PIL import Image
import pytesseract
import io
import spacy

nlp = spacy.load("en_core_web_sm")
app = Flask(__name__)

@app.route('/upload', methods=['POST'])
def upload():
    f = request.files['file']
    img = Image.open(f.stream).convert('RGB')
    text = pytesseract.image_to_string(img, lang='eng')
    doc = nlp(text)
    ents = [(ent.text, ent.label_) for ent in doc.ents]
    return jsonify({'text': text, 'entities': ents})
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

Run:

```bash
cd backend
python app.py
# Test with curl:
curl -F "file=@sample_scan.jpg" http://127.0.0.1:5000/upload
```

(For PDFs, you can use `pdf2image` to convert pages to images first.)

Day 2 — Small spatial processing: NDVI/NDWI (deliverable: water/veg GeoJSON)

- Download a Sentinel-2 scene (Copernicus) for a small village bounding box OR pull sample imagery from Bhuvan (see citations). Copernicus Data Space Ecosystem bhuvan.nrsc.gov.in
- Compute NDVI and NDWI using rasterio and numpy. Threshold to create binary masks for vegetation & water.
- Vectorize masks into GeoJSON (rasterio.features.shapes or `scikit-image` contours).
- Save GeoJSON to `data/water.geojson` and `data/veg.geojson`.

NDWI snippet (Sentinel-2 bands):

```python
import rasterio
import numpy as np
from rasterio.features import shapes
import json
from shapely.geometry import shape, mapping

# open green (B03) and nir (B08) bands (tifs)
g_ds = rasterio.open('B03.tif')
n_ds = rasterio.open('B08.tif')
g = g_ds.read(1).astype('float32')
n = n_ds.read(1).astype('float32')

ndwi = (g - n) / (g + n + 1e-8)
```

```
water_mask = ndwi > 0.1  # threshold: tune for your scene

# vectorize
results = (
    {'properties': {'ndwi': v}, 'geometry': s}
    for i, (s, v) in enumerate(
        shapes(water_mask.astype('uint8'), mask=None, transform=g_ds.transform))
)

features = [r for r in results if r['properties']['ndwi'] == 1]
geojson = {"type": "FeatureCollection", "features": features}
with open('data/water.geojson','w') as f:
    json.dump(geojson, f)
```

> Thresholds need tuning by scene — 0.1 is a starting point.

Day 3 — Shapefiles & DB integration (deliverable: FRA GeoJSON + DB ingest)

- If you have FRA shapefiles, convert to GeoJSON using `ogr2ogr` or `geopandas`.
- For demo, create a small synthetic FRA GeoJSON with a couple of polygons and attributes (name, patta_id, status).
- Ingest GeoJSON into PostGIS (or keep as file for MVP).

ogr2ogr example:

```bash
ogr2ogr -f "PostgreSQL" PG:"dbname=fra_db user=your_user" fra_polygons.shp -nln fra_patta
# or use geopandas.to_postgis in python
```

Day 4 — WebGIS frontend (deliverable: interactive Leaflet map)

- Create a simple `frontend/index.html` that:
  - Loads Leaflet base-layer (OpenStreetMap).
  - Calls backend endpoints `/api/fra`, `/api/water`, `/api/veg` returning GeoJSON.
  - Adds toggles (checkboxes) to show/hide layers and a simple filter by village name.

Minimal Leaflet + fetch example:

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>FRA Atlas Prototype</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
</head>
<body>
  <div id="map" style="height:90vh;"></div>
  <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
  <script>
    var map = L.map('map').setView([22.0, 80.0], 6);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(map);

    fetch('/api/water').then(r=>r.json()).then(data=>{
      L.geoJSON(data).addTo(map);
    });
    fetch('/api/fra').then(r=>r.json()).then(data=>{
      L.geoJSON(data,{onEachFeature: function(f,layer){
        layer.bindPopup(JSON.stringify(f.properties));
      }}).addTo(map);
    });
  </script>
</body>
</html>
```

- Serve `frontend` folder via Flask static routes for the prototype.

Day 5 — DSS rules engine + basic dashboard (deliverable: recommendations endpoint + dashboard)

- Create a small Python module `dss.py` with rule-based functions that take attributes and asset indices and return recommended schemes and priority.
- Expose `/api/recommendations?patta_id=...` which returns recommended schemes and rationale.
- Add a dashboard panel in the frontend showing top-5 recommended interventions (simple HTML list).

Example rule (pseudo):

```python
```

```python
def recommend(record, water_index):
    recs = []
    if water_index < 0.2 and record['land_size'] > 0.2:
        recs.append({'scheme':'Jal Shakti (borewell)','reason':'Low water index & farmland'})
    if record['landholding_type']=='smallholder' and record['is_farmer']:
        recs.append({'scheme':'PM-KISAN','reason':'Farmer with landholding'})
    return recs
```

Day 6 — Polish + demo + PPT export

- Test workflow end-to-end with sample inputs.
- Add simple logging, README with instructions to run.
- Prepare 3–5 minute demo script and record a short video or prepare screenshots.
- Export SIH PPT slides to PDF (fill content we built earlier).

---

5) Extra helpful commands & tips

- Convert PDF page to image:

  ```bash
  pip install pdf2image
  # then in Python use pdf2image.convert_from_path('file.pdf')
  ```

- Convert shapefile → GeoJSON:

  ```bash
  ogr2ogr -f GeoJSON out.geojson in.shp
  ```

- Use QGIS to inspect GeoJSON / shapefiles visually (optional; great for debugging).
- If rasterio install problems: ensure `gdal` is installed first (`brew install gdal` / `sudo apt install gdal-bin libgdal-dev`).

---

6) Recommended lightweight stack summary (for your README)

- Backend: Python + Flask
- DB: PostgreSQL + PostGIS (or SQLite for quick demo)
- OCR & NLP: Tesseract + spaCy
- Raster processing: rasterio + numpy + scikit-image
- Vector: geopandas + shapely
- Frontend: Leaflet (plain HTML + JS)
- Dev tools: VS Code, Git, Postman

Cited references for core tools & data sources: Python, PostGIS, Tesseract, Leaflet, Sentinel/Bhuvan. Python.org PostGIS Tesseract OCR leafletjs.com Copernicus Data Space Ecosystem bhuvan.nrsc.gov.in

---

7) Example repo structure & starter files I can create for you

If you want, I can auto-generate a starter repo containing:

- `backend/app.py` (Flask OCR + endpoints)
- `backend/dss.py` (simple rules engine)
- `frontend/index.html` (Leaflet sample)
- A `data/` folder with a tiny sample `fra.geojson` and a generated `water.geojson`
- `README.md` with exact run commands

Would you like me to create these files now and give you a zip / GitHub-ready repo? (If yes — I'll generate the starter code & ready-to-run files.)

That's everything you need to **start immediately** and hit a working prototype in 5–6 days. Want me to generate the starter repo + the working `app.py` + `index.html` now so you can run it locally?

yes

I've created a **ready-to-run starter repo** for your prototype. It includes:

- **backend/app.py** → Flask API (OCR upload, GeoJSON endpoints, serves frontend)
- **backend/dss.py** → Simple rule-based scheme recommendation engine

- **frontend/index.html** → Leaflet web map (layers: FRA claims, water, vegetation)
- **data/** → Sample GeoJSON files (FRA claim, pond, forest patch)
- **README.md** → Setup & run instructions

📦 Download it here: fra_prototype.zip

Do you want me to also show you **step-by-step how to run it locally** (terminal commands + testing in browser)?

yes

Perfect 👍 Here's your **step-by-step guide to run the prototype locally** after downloading `fra_prototype.zip`.

---

1. Unzip the project

Download the file and unzip it somewhere (say `~/projects/`).
It will create this structure:

```kotlin
fra_prototype/
  backend/
    app.py
    dss.py
  frontend/
    index.html
  data/
    fra.geojson
    water.geojson
    veg.geojson
  README.md
```

---

2. Create a virtual environment

Open your terminal and go inside the project:

```bash
cd fra_prototype
python3 -m venv venv
source venv/bin/activate
```

---

3. Install dependencies

```bash
pip install --upgrade pip
pip install flask pillow pytesseract spacy
python -m spacy download en_core_web_sm
```

👉 Also install **Tesseract OCR engine** on your system:

- macOS: `brew install tesseract`
- Ubuntu: `sudo apt install tesseract-ocr`

(Windows → download from: https://github.com/UB-Mannheim/tesseract/wiki ↗)

---

4. Run the backend

```bash
cd backend
python app.py
```

You should see:

```csharp
* Running on http://127.0.0.1:5000
```

---

5. Open the web app

Open your browser and go to:

👉 http://127.0.0.1:5000 ↗

You'll see a **Leaflet map**:

- Blue polygon = water body
- Green polygon = vegetation/forest
- Polygon with popup = FRA claim

---

6. Test OCR endpoint

In another terminal (while backend is running), test with an image:

```bash
curl -F "file=@../data/sample.jpg" http://127.0.0.1:5000/upload
```

(Replace `sample.jpg` with any scanned/typed image you have).

You'll get back **OCR text + named entities**.

---

7. Try DSS rules engine

```bash
cd backend
python dss.py
```

It will print demo recommendations like:

```bash
[{'scheme': 'Jal Shakti (borewell)', 'reason': 'Low water index & farmland'},
 {'scheme': 'PM-KISAN', 'reason': 'Farmer with small landholding'}]
```

✅ That's it — you now have a **working prototype**:

- OCR pipeline
- FRA + asset layers on map
- Rule-based DSS

Do you also want me to show you **how to deploy this quickly** (e.g., on free services like **Render / Railway / Heroku**) so you can share a live demo link?

what exactly it is doing?

Great question 🙌 Let's break down **exactly what your current prototype is doing** step by step.

🔹 1. OCR + Entity Extraction

- You can upload a scanned **FRA document (claim/patta/verification)** to the `/upload` endpoint.
- The backend uses:
  - **Tesseract OCR** → extracts raw text from the scanned image.
  - **spaCy NLP** → detects key entities (like names, places, numbers, etc.).
- Output: JSON with extracted text + recognized entities.
  👉 This simulates digitization of **non-digital FRA records**.

🔹 2. FRA Claim Data (Sample GeoJSON)

- A **sample FRA polygon** (GeoJSON) is stored in `/data/fra.geojson`.
- When you open the web app, this polygon is drawn on the map with a popup showing:
  - Patta holder's name
  - Claim status
    👉 This represents **digitized FRA claims mapped to real-world coordinates**.

🔹 3. Asset Mapping (Demo: Water & Vegetation)

- Two sample **GeoJSON layers** are included:
  - `/data/water.geojson` → a pond (blue polygon)
  - `/data/veg.geojson` → a forest patch (green polygon)
- These are displayed as separate layers on the Leaflet map.
  👉 This simulates **AI/ML-based detection of natural assets from satellite imagery** (in reality you'd run NDWI/NDVI on Sentinel/Bhuvan data).

🔹 4. WebGIS Frontend

- The **Leaflet map** loads:
  - FRA claims layer
  - Water bodies layer
  - Vegetation layer
- You can click polygons to see info (patta holder, type of asset, etc.).
  👉 This provides the **interactive FRA Atlas** where officials/users can visualize claims + assets.

🔹 5. Decision Support System (DSS) — Rule-Based Demo

- `dss.py` contains simple rules:
  - If **low water index** + farmland → Recommend *Jal Shakti (borewell)*
  - If **smallholder farmer** → Recommend *PM-KISAN*
- Running it shows a **list of recommended schemes** based on attributes.
  👉 This demonstrates how the DSS would **suggest schemes for each FRA holder**.

🔹 So in short, your prototype:

1. **Digitizes FRA records** (OCR + entity extraction).
2. **Maps FRA claims** (GeoJSON polygons).
3. **Overlays asset layers** (water, vegetation).
4. Provides a **WebGIS portal** to view all layers.
5. Runs a **basic DSS engine** for scheme recommendations.

⚡ Right now it's a **mock/demo** — you're showing the flow with sample data. In the final version, you'd:

- Replace sample GeoJSON with **real shapefiles/FRA data**.
- Replace demo water/veg polygons with **AI-processed satellite outputs**.
- Expand the DSS with **more schemes + rules/ML models**.

👉 Do you want me to now help you **extend this demo** so that instead of static sample water/veg polygons, you actually compute them from a free Sentinel-2 image (NDVI/NDWI workflow)? That way your prototype will look more "AI-powered" instead of just mock data.

yes

Awesome 🚀 Let's upgrade your prototype so it **actually computes water and vegetation layers from real satellite data** instead of mock polygons. This will make your demo look truly **AI/remote-sensing powered**.

🔹 Step 1: Get Sample Sentinel-2 Data

You can get **free Sentinel-2 imagery** (bands in `.tif` format):

1. **Copernicus Open Access Hub** → scihub.copernicus.eu ↗ (requires login).
2. **USGS EarthExplorer** → earthexplorer.usgs.gov ↗.
3. **Sentinel Hub Playground** → apps.sentinel-hub.com/sentinel-playground ↗ (quick sample downloads).

👉 For a quick demo, download **3 bands** from Sentinel-2 (same area, same date):

- **B03 (Green)** → needed for NDWI
- **B04 (Red)** → needed for NDVI
- **B08 (NIR)** → needed for NDVI & NDWI

🔹 Step 2: Install Required Libraries

Make sure these are installed in your venv:

```bash
pip install rasterio numpy geopandas shapely
```

🔹 Step 3: Write Processing Script

Create `backend/process_satellite.py`:

python

```python
import rasterio
import numpy as np
import json
from rasterio.features import shapes

def compute_index(band1_path, band2_path, index="NDVI"):
    with rasterio.open(band1_path) as b1, rasterio.open(band2_path) as b2:
        band1 = b1.read(1).astype("float32")
        band2 = b2.read(1).astype("float32")
        profile = b1.profile
    if index == "NDVI":
        return (band2 - band1) / (band2 + band1 + 1e-6), profile
    elif index == "NDWI":
        return (band1 - band2) / (band1 + band2 + 1e-6), profile
    else:
        raise ValueError("Unknown index")

def raster_to_geojson(mask, transform, threshold=0.2):
    results = (
        {"properties": {"class": int(v)}, "geometry": s}
        for s, v in shapes(mask.astype("uint8"), transform=transform)
    )
    features = [r for r in results if r["properties"]["class"] == 1]
    return {"type": "FeatureCollection", "features": features}

if __name__ == "__main__":
    # Example: NDVI (Red=B04, NIR=B08)
    ndvi, profile = compute_index("../data/B04.tif", "../data/B08.tif", index="NDVI")
    ndvi_mask = ndvi > 0.3  # vegetation threshold
    ndvi_geojson = raster_to_geojson(ndvi_mask, profile["transform"])
    with open("../data/veg.geojson", "w") as f:
        json.dump(ndvi_geojson, f)

    # Example: NDWI (Green=B03, NIR=B08)
    ndwi, profile = compute_index("../data/B03.tif", "../data/B08.tif", index="NDWI")
    ndwi_mask = ndwi > 0.2  # water threshold
    ndwi_geojson = raster_to_geojson(ndwi_mask, profile["transform"])
    with open("../data/water.geojson", "w") as f:
        json.dump(ndwi_geojson, f)
```

```
print("GeoJSONs generated: veg.geojson & water.geojson")
```

🔹 Step 4: Run the Processor

1. Place your Sentinel-2 `.tif` files (B03, B04, B08) into the `data/` folder. Example filenames: `B03.tif`, `B04.tif`, `B08.tif`.
2. Run:

```bash
cd backend
python process_satellite.py
```

3. This will overwrite `veg.geojson` and `water.geojson` with **real polygons** detected from satellite imagery.

🔹 Step 5: View in Web Map

Restart your Flask app:

```bash
python app.py
```

Go to http://127.0.0.1:5000 ↗.

- Green = vegetation (from NDVI)
- Blue = water (from NDWI)
- FRA claim polygon still there

👉 Now you can say in your demo:

> "This map is generated from **actual Sentinel-2 data** — vegetation is detected using NDVI, water using NDWI — and combined with FRA claims to form an **AI-powered FRA Atlas prototype**."

⚡ Next step we can do (if you like): add a **basic dashboard** page in the frontend (tables/charts) to show FRA stats + DSS recommendations.

Do you want me to extend the frontend with a **dashboard panel** (side menu showing extracted OCR data + DSS recommendations) so it looks more like a decision support system?

yes

Perfect 👍 Let's extend your **frontend** so it feels more like a real **Decision Support Dashboard**, not just a map.

We'll add a **side panel** next to the map that shows:

1. **OCR extracted data** (from uploaded FRA docs)
2. **Scheme recommendations** (from DSS engine)
3. **Basic stats** (number of FRA claims, water/veg area detected)

🔹 Step 1: Update Frontend Layout

Replace `frontend/index.html` with this:

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>FRA Atlas Prototype</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
  <style>
    body, html {margin:0; padding:0; height:100%; display:flex;}
    #sidebar {width:30%; padding:10px; background:#f4f4f4; overflow-y:auto;}
    #map {flex:1; height:100%;}
    h2 {margin-top:0;}
    .card {background:white; padding:10px; margin:10px 0; border-radius:8px; box-shadow:0
2px 4px rgba(0,0,0,0.1);}
  </style>
</head>
<body>
  <div id="sidebar">
    <h2>FRA DSS Dashboard</h2>
```

```
    <div class="card">
      <h3>Upload FRA Document</h3>
      <input type="file" id="docInput"/>
      <button onclick="uploadDoc()">Process</button>
      <pre id="ocrOutput"></pre>
    </div>

    <div class="card">
      <h3>Scheme Recommendations</h3>
      <button onclick="getRecs()">Get Recommendations</button>
      <ul id="recList"></ul>
    </div>

    <div class="card">
      <h3>Stats</h3>
      <p id="stats">Loading...</p>
    </div>
  </div>

  <div id="map"></div>

  <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
  <script>
    var map = L.map('map').setView([22.0, 80.0], 5);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(map);

    let fraCount = 0, waterCount = 0, vegCount = 0;

    fetch('/api/fra').then(r=>r.json()).then(data=>{
      fraCount = data.features.length;
      L.geoJSON(data,{onEachFeature:(f,l)=>l.bindPopup("Patta Holder:
"+f.properties.name)}).addTo(map);
      updateStats();
    });

    fetch('/api/water').then(r=>r.json()).then(data=>{
      waterCount = data.features.length;
      L.geoJSON(data,{style:{color:'blue'}}).addTo(map);
      updateStats();
    });
```

```
    fetch('/api/veg').then(r=>r.json()).then(data=>{
      vegCount = data.features.length;
      L.geoJSON(data,{style:{color:'green'}}).addTo(map);
      updateStats();
    });

    function updateStats() {
      document.getElementById("stats").innerText =
        `FRA Claims: ${fraCount}\nWater Bodies: ${waterCount}\nVegetation Areas: ${vegCount}`;
    }

    function uploadDoc() {
      let file = document.getElementById("docInput").files[0];
      let formData = new FormData();
      formData.append("file", file);
      fetch('/upload',{method:'POST', body:formData})
        .then(r=>r.json())
        .then(data=>{
          document.getElementById("ocrOutput").innerText = data.text + "\\nEntities: " + JSON.stringify(data.entities,null,2);
        });
    }

    function getRecs() {
      fetch('/api/recommendations').then(r=>r.json()).then(data=>{
        let list = document.getElementById("recList");
        list.innerHTML = "";
        data.forEach(r=>{
          let li = document.createElement("li");
          li.innerText = r.scheme + " → " + r.reason;
          list.appendChild(li);
        });
      });
    }
  </script>
</body>
</html>
```

🔹 Step 2: Add Recommendations API in Backend

Edit `backend/app.py`, add this endpoint **below other routes**:

```python
from dss import recommend

@app.route('/api/recommendations')
def recommendations():
    # Demo record (would come from DB in real system)
    record = {"land_size":0.5, "landholding_type":"smallholder", "is_farmer":True}
    water_index = 0.1  # demo value
    recs = recommend(record, water_index)
    return jsonify(recs)
```

🔹 Step 3: Run & Test

1. Restart Flask app:

   ```bash
   cd backend
   python app.py
   ```

2. Open http://127.0.0.1:5000 ↗.
   You'll now see:
   - **Left sidebar dashboard**
   - Upload FRA document → see OCR text + entities
   - Click "Get Recommendations" → see DSS schemes
   - Stats card showing counts of FRA claims, water bodies, vegetation

---

✅ Now your prototype looks like a **real Decision Support System**:

- Map visualization (Atlas)
- OCR digitization (records)
- AI asset mapping (vegetation/water)
- DSS recommendations (schemes)
- Dashboard summary

Do you want me to also add a **filter option in the sidebar** (e.g., select state/district/tribal group) so you can simulate FRA progress tracking?

yes

Perfect ✅ Let's add **filters (state/district/tribal group)** in your dashboard so users can interactively filter FRA claims and simulate **progress tracking**.

🔹 Step 1: Update `frontend/index.html`

Replace the **Stats card** with a **Filters + Stats card**:

```html
<div class="card">
  <h3>Filters</h3>
  <label>State:
    <select id="stateFilter" onchange="applyFilters()">
      <option value="">All</option>
      <option value="Madhya Pradesh">Madhya Pradesh</option>
      <option value="Odisha">Odisha</option>
      <option value="Tripura">Tripura</option>
      <option value="Telangana">Telangana</option>
    </select>
  </label><br><br>
  <label>Tribal Group:
    <select id="tribeFilter" onchange="applyFilters()">
      <option value="">All</option>
      <option value="Gond">Gond</option>
      <option value="Santhal">Santhal</option>
      <option value="Khasi">Khasi</option>
    </select>
  </label>
</div>

<div class="card">
  <h3>Stats</h3>
  <p id="stats">Loading...</p>
</div>
```

🔹 Step 2: Add Filtering Logic in Frontend JS

Modify your `fetch('/api/fra')` part:

```javascript
let fraLayer;
let fraData;

fetch('/api/fra').then(r=>r.json()).then(data=>{
  fraData = data;
  fraLayer = L.geoJSON(data,{
    onEachFeature:(f,l)=>l.bindPopup("Patta Holder: "+f.properties.name)
  }).addTo(map);
  fraCount = data.features.length;
  updateStats();
});

function applyFilters() {
  let stateVal = document.getElementById("stateFilter").value;
  let tribeVal = document.getElementById("tribeFilter").value;

  if (fraLayer) {
    map.removeLayer(fraLayer);
  }

  let filtered = {
    type: "FeatureCollection",
    features: fraData.features.filter(f=>{
      let matchState = !stateVal || f.properties.state === stateVal;
      let matchTribe = !tribeVal || f.properties.tribe === tribeVal;
      return matchState && matchTribe;
    })
  };

  fraCount = filtered.features.length;
  fraLayer = L.geoJSON(filtered,{
    onEachFeature:(f,l)=>l.bindPopup(
      `Patta Holder: ${f.properties.name}<br>State: ${f.properties.state}<br>Tribe:
${f.properties.tribe}`
    )
  }).addTo(map);
```

```
  updateStats();
}
```

🔹 Step 3: Update FRA GeoJSON Sample Data

Open `data/fra.geojson` and add **state + tribal group properties**:

```json
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "patta_id": 1,
        "name": "Ram Das",
        "status": "Granted",
        "state": "Madhya Pradesh",
        "tribe": "Gond"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[80,22],[80.1,22],[80.1,22.1],[80,22.1],[80,22]]]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "patta_id": 2,
        "name": "Sita Devi",
        "status": "Pending",
        "state": "Odisha",
        "tribe": "Santhal"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[80.2,22.2],[80.3,22.2],[80.3,22.3],[80.2,22.3],[80.2,22.2]]]
      }
    }
```

```
  ]
}
```

🔹 Step 4: Restart Flask App

```bash
cd backend
python app.py
```

Open http://127.0.0.1:5000 ↗.

Now you can:

- Filter by **State** (Madhya Pradesh, Odisha, etc.)
- Filter by **Tribal Group** (Gond, Santhal, Khasi)
- Map updates to show only matching FRA claims
- Stats panel updates FRA claim counts

✅ Now your prototype has:

- **OCR digitization**
- **AI asset mapping** (NDVI/NDWI from Sentinel-2)
- **WebGIS Atlas**
- **DSS recommendations**
- **Filters for state/tribe** → simulates **progress tracking & policy targeting**

👉 Do you want me to also add a **progress bar / chart (like % of granted vs pending claims)** in the dashboard for extra impact in your demo?

generate a readme file for this project, which explains entire process, and workflow of the project, the tools used, requirement, etc which are necessary for readme files.

Here's a **complete** `README.md` you can drop directly into your project folder.
I've written it so it looks professional, explains the **workflow**, **tools**, **requirements**, and how to **run** the prototype.

---

markdown

```markdown
# 🌳 AI-Powered FRA Atlas & WebGIS DSS Prototype

## 📌 Overview
This project is a working prototype for the **AI-powered FRA Atlas and WebGIS-based Decision Support System (DSS)**.
It demonstrates how **Forest Rights Act (FRA) claims** can be **digitized, mapped, and analyzed** alongside **AI-driven asset detection** (water, vegetation) to support decision-making and scheme allocation.

The prototype integrates:
- **OCR digitization of FRA documents**
- **AI-based asset mapping from satellite data (NDVI & NDWI)**
- **WebGIS visualization with filters**
- **Rule-based Decision Support System (DSS)** for recommending schemes
- **Dashboard** with stats, OCR results, scheme recommendations, and filters

---

## ⚙️ Workflow

### 1. **Data Collection & Digitization**
- FRA claim documents (PDF/JPEG) are uploaded.
- OCR (**Tesseract**) extracts text.
- NLP (**spaCy**) detects entities (patta holder, village, etc.).
- Records are standardized and stored.

### 2. **AI-Powered FRA Atlas**
- Sentinel-2 satellite bands processed using **Rasterio + Numpy**:
  - **NDVI** → detects vegetation/forest cover.
  - **NDWI** → detects water bodies (ponds, rivers).
- Binary masks are vectorized into **GeoJSON layers**.

### 3. **WebGIS Integration**
- **LeafletJS** frontend shows interactive layers:
```

  - FRA claims (IFR/CR/CFR)
  - Water bodies
  - Vegetation/forests
- Filters: by **State** and **Tribal Group**.
- Stats panel shows counts of claims and assets.

4. **Decision Support System (DSS)**
- Simple **rule-based engine** links FRA attributes + asset indices to government schemes.
- Example rules:
  - Low water availability → Recommend **Jal Shakti (borewell)**.
  - Smallholder farmer → Recommend **PM-KISAN**.
- Future scope: AI/ML models for smarter recommendations.

5. **Dashboard**
- **Upload section** → process FRA docs.
- **Recommendations panel** → DSS outputs.
- **Stats & Filters** → progress tracking by state/tribe.
- **Interactive map** → visualize assets and claims.

🛠️ Tools & Frameworks

🔹 Backend
- [Python 3](https://www.python.org/)
- [Flask](https://flask.palletsprojects.com/) – API & server
- [Tesseract OCR](https://github.com/tesseract-ocr/tesseract) – text extraction
- [spaCy](https://spacy.io/) – NLP for entity recognition
- [Rasterio](https://rasterio.readthedocs.io/) – satellite raster processing
- [Numpy](https://numpy.org/) – NDVI/NDWI calculations
- [GeoPandas](https://geopandas.org/) – vector data handling
- [Shapely](https://shapely.readthedocs.io/) – geometry operations

🔹 Frontend
- [LeafletJS](https://leafletjs.com/) – interactive maps
- HTML/CSS/JavaScript – dashboard & UI

🔹 Database (Optional, for scaling)
- [PostgreSQL + PostGIS](https://postgis.net/) – spatial DB for FRA claims & assets

---

````
## 📦 Requirements

### System Dependencies
- Python 3.11+
- Tesseract OCR
  - macOS: `brew install tesseract`
  - Ubuntu: `sudo apt install tesseract-ocr`
  - Windows: [Download here](https://github.com/UB-Mannheim/tesseract/wiki)

### Python Libraries
```bash
pip install flask pillow pytesseract spacy rasterio geopandas shapely numpy
python -m spacy download en_core_web_sm
````

▶ Running the Prototype

1. **Clone / Unzip the Project**

```bash
cd fra_prototype
```

2. **Setup Virtual Environment**

```bash
python3 -m venv venv
source venv/bin/activate
```

3. **Install Dependencies**

```bash
pip install --upgrade pip
pip install flask pillow pytesseract spacy rasterio geopandas shapely numpy
python -m spacy download en_core_web_sm
```

4. **Run Backend**

```bash
```
```

```
cd backend
python app.py
```

Server runs on: http://127.0.0.1:5000 ↗

5. **Open Dashboard**
    - Open your browser at http://127.0.0.1:5000 ↗
    - Map + Dashboard will load.
    - Try uploading a scanned FRA doc (JPG/PDF).
    - View assets + DSS recommendations.

🌍 Folder Structure

```bash
fra_prototype/
│── backend/
│   ├── app.py          # Flask server + APIs
│   ├── dss.py          # Rule-based DSS engine
│   ├── process_satellite.py # NDVI/NDWI generator
│── frontend/
│   ├── index.html        # WebGIS dashboard
│── data/
│   ├── fra.geojson       # FRA sample claims
│   ├── water.geojson      # Water bodies (from NDWI)
│   ├── veg.geojson       # Vegetation (from NDVI)
│── README.md           # Documentation
```

📊 Future Scope

- Integrate **real FRA shapefiles** (state govt datasets).
- Automate **Sentinel-2/Bhuvan API integration**.
- Expand DSS with **ML models** (not just rules).
- Add **progress charts** (granted vs pending claims).
- Enable **mobile app reporting** by patta holders.
- Deploy on **cloud (AWS / GCP / Render)** for real-time access.

👥 Target Users

- **Ministry of Tribal Affairs**
- **Tribal Welfare Departments** (State/District)
- **Forest & Revenue Departments**
- **Planning Authorities**
- **NGOs working with tribal communities**

📚 References

- Forest Rights Act, 2006 (PDF) ↗
- NRSC Bhuvan Portal ↗
- FAO Land Cover Classification ↗
- MoTA Annual FRA Reports ↗
- PM Gati Shakti Master Plan ↗

✨ This prototype shows the **end-to-end workflow** of an **AI + GIS powered FRA Atlas**: from **digitization → mapping → DSS → dashboard**. It can be scaled into a production-ready solution with real datasets.

```yaml


---

Would you like me to also generate a **diagram (infographic-style workflow chart)** to include in the README so it looks visually appealing on GitHub?
```

with refrence to my chat,tell me how to use WebGIS , how it's gonna help me? what role will it serve?

Excellent question 🙌 — WebGIS is one of the **core pillars** of your FRA Atlas prototype. Let me explain it in the context of your project and workflow.

🌍 What is WebGIS?

- **WebGIS** = **Geographic Information System on the Web**.
- It allows you to **store, visualize, analyze, and share spatial data** interactively through a browser, without needing heavy desktop software (like QGIS or ArcGIS).
- Typically uses:
  - **Backend GIS server** (GeoServer, MapServer, or custom Flask/PostGIS APIs).
  - **Database with spatial support** (PostgreSQL + PostGIS).
  - **Frontend mapping libraries** (Leaflet, Mapbox, OpenLayers).

🔹 How WebGIS fits into your FRA Atlas

Think of WebGIS as the **"front window"** of your project — the place where all digitized, processed, and AI-generated outputs are made **visible, explorable, and usable**.

Here's its role step by step:

1. Integration of Layers

- WebGIS allows you to combine **multiple spatial layers**:
  - FRA claims (IFR, CR, CFR shapefiles → GeoJSON/PostGIS)
  - AI asset maps (vegetation/NDVI, water/NDWI, homesteads)
  - Socio-economic overlays (groundwater index, scheme locations, infrastructure from PM Gati Shakti)
- Without WebGIS, these layers would remain as static files.
- With WebGIS, users **toggle, filter, overlay, and compare layers interactively**.

---

2. Visualization of FRA Atlas

- FRA Atlas = **digital, interactive map** of claims + resources.
- WebGIS ensures:
  - Officials can zoom into **village → district → state** levels.
  - Tribal groups can see **which claims are pending vs granted**.
  - NGOs/Planners can view **asset distribution** (e.g., water scarcity zones).

---

3. Progress Tracking

- WebGIS supports **filters** (state, district, tribal group).
- In your prototype:
  - Selecting *Odisha* shows only Odisha FRA claims.
  - Filtering *Gond tribe* shows tribal-specific claim progress.
- Helps govt officials in **real-time monitoring & decision-making**.

---

4. Decision Support System (DSS) Integration

- DSS outputs (like "Recommend Jal Shakti in low water villages") can be:
  - Shown as **map annotations** (pinpoints where schemes should go).
  - Visualized in **dashboards linked to maps**.
- Example: If NDWI shows low water → map highlights villages in red → DSS suggests *borewell intervention*.
- WebGIS makes DSS **spatially actionable**.

---

5. Accessibility & Transparency

- A **WebGIS portal** can be accessed by:
  - **Government ministries** → for monitoring & reporting.
  - **Local officers** → for planning field interventions.
  - **NGOs & communities** → for awareness & advocacy.
- Ensures **transparency** in FRA implementation (vs opaque paper records).

🔑 Why WebGIS is crucial for your project

- **Bridges data → decisions**: AI models output shapefiles/GeoJSON → WebGIS makes them actionable.
- **Supports multi-level governance**: from village to state, all can access the same portal.
- **Visual storytelling**: instead of raw data, decision-makers see **where gaps exist** (pending claims, lack of water).
- **Scalability**: Start with 4 states → later scale to entire India by just adding new layers.

🚀 In your prototype

- You're using **LeafletJS (WebGIS frontend)** + **Flask backend APIs** to serve FRA + asset layers.
- This allows you to **demonstrate how WebGIS makes FRA data interactive**:
  - Show **FRA claim polygons**.
  - Overlay **vegetation/water detection**.
  - Use **filters (state/tribe)** to track progress.
  - Display **DSS recommendations spatially**.

👉 So WebGIS in your project is **not just a map** — it's the **central hub** where all the digitization, AI asset mapping, and DSS logic **come together in one interface** for decision-makers and communities.

Do you want me to also **compare different WebGIS stacks** (e.g., simple Leaflet + Flask vs. full GeoServer + PostGIS) so you can decide what's best for a **5-day prototype vs. future production system**?

tell me step by step process to use it and apply it.

Awesome — here's a **clear, step-by-step playbook** for using and applying the WebGIS part of your FRA Atlas prototype. I'll give you two tracks:

- **Track A (Prototype Now)**: Use the repo we made (Flask + Leaflet + GeoJSON files).
- **Track B (Level Up Later)**: Swap files for a spatial database + GeoServer when you're ready.

Track A — Prototype Now (Leaflet + Flask + GeoJSON)

0) One-time setup

1. Unzip `fra_prototype.zip`.
2. Create and activate a Python venv, then install deps:

```bash
```

```
cd fra_prototype
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install flask pillow pytesseract spacy rasterio geopandas shapely numpy
python -m spacy download en_core_web_sm
```

3. (If you'll demo OCR) Install **Tesseract** on your machine (brew/apt/Windows installer).

---

1) Start the WebGIS

```bash
cd backend
python app.py
```

Open the app at http://127.0.0.1:5000 ↗
You'll see:

- **Map** on the right (Leaflet)
- **Dashboard** on the left with **Upload**, **Recommendations**, **Filters**, **Stats**

---

2) Use the map (what to click / see)

- The map loads **three layers**:
  - **FRA claims** (`/api/fra`) → polygons with popups (patta holder, status, etc.)
  - **Water** (`/api/water`) → blue polygons (from NDWI or demo)
  - **Vegetation/forest** (`/api/veg`) → green polygons (from NDVI or demo)
- Click a polygon → see its properties.

Tip: The **Stats** card shows how many features are currently visible. This updates when you filter.

---

3) Use filters for progress tracking

On the left:

- **State** dropdown

- **Tribal Group** dropdown

Choose a state or tribe → the **FRA layer filters** to matching features.
Map and **Stats** update instantly.
This is how you'll **demonstrate monitoring at State/District/Tribe granularity**.

(If you need district-level soon: add a `district` property to features and another dropdown in the sidebar, then reuse the same `applyFilters()` logic.)

---

4) Run real asset mapping (NDVI/NDWI) and refresh map

1. Put your Sentinel-2 TIFs into `data/` (B03, B04, B08 for the same date & area).
2. Generate real GeoJSON:

   ```bash
   cd backend
   python process_satellite.py
   ```

   This **overwrites** `data/water.geojson` and `data/veg.geojson` with **real detections**.
3. Refresh the browser — you'll now see **actual vegetation & water** overlays derived from satellite data.

Demo script line: "These green and blue polygons come from **NDVI/NDWI computed on Sentinel-2** for this region."

---

5) Try the OCR digitization

1. In the **Upload FRA Document** card, choose a scanned image (JPG/PNG).
2. Click **Process**.
   You'll see:
   - **Extracted text**
   - **Named entities** (names/places/dates detected by spaCy)

What this proves: You can digitize legacy FRA docs and extract structured info for the atlas/database.

---

6) See DSS recommendations

Click **Get Recommendations** in the **Scheme Recommendations** card.

You'll see a list like:

- "Jal Shakti (borewell) → Low water index & farmland"
- "PM-KISAN → Farmer with small landholding"

What this proves: The WebGIS is not just a map — it's a **Decision Support System** with **spatially anchored** recommendations.

---

7) Add your own FRA shapefiles (no code changes needed)

If you have a shapefile:

1. Convert to GeoJSON:

   ```bash
   ogr2ogr -f GeoJSON fra.geojson your_shapefile.shp
   ```

2. Replace `data/fra.geojson` with your new file.
3. Reload browser → Your **real claims** appear with popups.

Schema tips (properties you should include):

```json
{
  "patta_id": "string/number",
  "name": "patta holder name",
  "status": "Granted|Pending|Rejected",
  "state": "Madhya Pradesh|Odisha|Tripura|Telangana",
  "tribe": "Gond|Santhal|Khasi|..."
}
```

---

8) Add more layers (groundwater, roads, scheme works)

- Put a new GeoJSON in `data/`, e.g. `groundwater.geojson`.
- Add a new Flask endpoint in `backend/app.py`:

  ```python
  @app.route('/api/groundwater')
  def get_groundwater():
  ```

```
    with open("../data/groundwater.geojson") as f:
        return jsonify(json.load(f))
```

- In `frontend/index.html`, fetch and add it with `L.geoJSON(...)`.
- Optionally style it and bind popups.

---

9) Presenting & storytelling (how to apply it in your demo)

- Start at **State view** → toggle filters to show target states (MP/Tripura/Odisha/Telangana).
- Zoom into a **district/village** with **low water** polygons → click recommendation button → highlight "Jal Shakti" suggestion.
- Upload an example **FRA doc** → show digitized text & entities.
- Conclude with: "This is an **integrated WebGIS**: digitization + AI asset mapping + DSS → **actionable scheme planning**."

Track B — Level Up Later (PostGIS + GeoServer)

When you need multi-user editing, big datasets, and fast rendering:

1) Move GeoJSON → PostGIS

- Install PostgreSQL + PostGIS.
- Load layers:

```bash
shp2pgsql -I your_shapefile.shp public.fra_claims | psql -d fra_db
```

- Or with GeoPandas:

```python
gdf.to_postgis("fra_claims", engine, if_exists="replace")
```

2) Publish layers with GeoServer

- Install GeoServer.
- Connect a **PostGIS datastore**.
- Publish `fra_claims`, `water`, `veg` as **WMS/WFS**.
- Style with SLD for **Pending/Granted** symbology.

3) Frontend swap

- Replace `fetch('/api/fra')` with WFS/WMS URLs.
- For WMS tiles:

```js
L.tileLayer.wms('http://<geoserver>/geoserver/wms', {
  layers: 'workspace:fra_claims',
  format: 'image/png',
  transparent: true
}).addTo(map);
```

4) Why do this

- Faster maps, huge data, styling in server, consistent access control, easier integration with other GIS tools.

Common Pitfalls & Quick Fixes

- **Rasterio install errors** → ensure **GDAL** is installed first (mac: `brew install gdal`, ubuntu: `sudo apt install gdal-bin libgdal-dev`).
- **Projection mismatch** → keep all layers in **EPSG:4326 (WGS84)** for the prototype.
- **Large GeoJSON slow** → simplify geometry (e.g., `mapshaper`) or move to PostGIS/GeoServer.
- **OCR gibberish** → use clearer scans; for Hindi/other scripts add Tesseract language packs.

Your “Apply It” Checklist (Prototype to Pilot)

1. **Replace sample GeoJSON** with **real FRA claims** (start with 1 block per state).
2. **Run NDVI/NDWI** for the same area/date → generate real water/veg layers.
3. **Add groundwater/infra** overlays (CSV/GeoJSON from state portals or PM Gati Shakti exports).
4. **Expand DSS rules** for 3–5 schemes with clear eligibility fields.
5. **Package a 3–5 min demo**: filters → assets → DSS → OCR → story of “targeted scheme allocation”.

If you want, I can also **add a small chart** (granted vs pending) in the sidebar and a **download CSV** button for filtered results — both are quick wins that impress judges.